Thomas ROSSAT-MIGNOD

Capture de mouvement pour améliorer le jeu des guitaristes

Stage effectué du 26 mars 2007 au 15 juin 2007

à l'Université de Napier
à Edimbourg en Ecosse

dans le cadre de la préparation d'un DUT Informatique

IUT du Limousin – Département Informatique

Remerciements

Je tiens à remercier **Grégory Leplâtre**, mon maître de stage, qui a su me guider dans l'élaboration de mon projet, et qui m'a permis d'effectuer mon stage dans de très bonnes conditions.

Je remercie également **Laurent Bourdier** et **Isabelle Gauchet**, qui, grâce au service des Relations Internationales mis en place cette année, m'ont aidé à trouver ce stage.

Résumé

J'ai effectué un stage à l'Université de Napier, à Edimbourg en Ecosse. Il s'agit d'un stage liant étroitement le traitement du son avec des mouvements. Ces mouvements effectués par des musiciens, peuvent être des mouvements usuels dans la musique, que nous pourrions même qualifier de naturels.

Pour réaliser ce projet, j'ai appris à me servir du logiciel Max/MSP, utile pour les traitements de messages MIDI et de signaux sonores. Une étude a dû être effectuée pour comprendre également le comportement de capteurs de mouvement. Quelques effets ont été implémentés (Wah-Wah, larsen, délai, chorus...) puis par la suite liés à certains mouvements effectués par un guitariste. Enfin, une interface graphique a par la suite été implémentée afin que n'importe quel guitariste puisse utiliser mon travail.

Le travail réalisé sera par la suite continué par d'autres personnes afin de le présenter à l'évènement NIME (New Interfaces for Musical Expression).

Table des matières

Introduction

J'ai effectué mon stage à l'Université de Napier à Edimbourg, capitale de l'Ecosse. Cette Université possède des centres de recherche dans tous les domaines : la biologie, la santé, les arts, les sciences sociales, le commerce et évidemment l'informatique. Ces centres de recherche sont très actifs et certains d'entre eux ont même obtenu des récompenses pour leurs travaux. C'est donc dans le centre de recherche sur l'intéraction homme-machine que mon stage s'est déroulé.

Dans ce centre de recherche, les principales études menées traitent de l'amélioration des interfaces homme-machine pour les handicapés. Mais des recherches concernant le son et les intéractions entre l'homme, la machine et le son sont également menées. C'est donc dans ce domaine que s'inscrit mon sujet de stage.

Le but du sujet de stage est de trouver un nouveau moyen pour les guitaristes d'améliorer leur jeu. Ces recherches sont basées sur l'étude des mouvements usuels. L'idée est d'attribuer à un mouvement un effet que le guitariste, en prenant l'habitude de jouer avec l'application, pourra maîtriser, et ainsi inventer un nouveau style de jeu de guitare.

Le travail que j'ai effectué doit être continué par la suite. Il représente en fait le début d'une recherche qui devrait donner lieu à une éventuelle présentation du produit à NIME (Nouvelles Interfaces pour l'Expression Musicale), événement majeur dans le domaine des intéractions homme-machine et son. Normalement, le travail devrait être terminé pour la prochaine édition de NIME en 2008.

Le travail s'est divisé en plusieurs étapes : dans un premier temps, une étude des travaux déjà effectués dans le cadre de NIME a dû être faite : cette étude était indispensable pour comprendre le contexte du sujet. Ensuite, il m'a fallu apprendre à utiliser le logiciel Max/MSP, logiciel de programmation visuelle du traitement du son, sur lequel repose tout le bon fonctionnement de mon projet. Ce n'est qu'après que l'analyse des capteurs de mouvement, et de leur comportement a été menée. L'élaboration d'algorithmes (en JavaScript) permettant d'exploiter les données renvoyées par les capteurs a représenté par la suite la majeure partie du travail. Enfin, une interface graphique a pu être développée en Java, pour une utilisation plus aisée de l'application.

Tout d'abord, je vous présenterai l'Université de Napier et le contexte du stage. Ensuite, je vous expliquerai les notions aquises indispensables à la bonne compréhension du travail effectué. Ce n'est que dans la dernière partie que je vous détaillerai le travail réalisé tout au long du stage.

1 L'université de Napier

1.1 Napier University

1.1.1 Historique de Napier University

Napier University, située à Edimbourg en Ecosse, est une institution dont le but est de proposer des cursus correspondant aux besoins des étudiants et des employeurs actuels. Napier a ouvert ses portes en 1964, sous le nom de Napier Technical College (Institut technique de Napier), au campus de Merchiston : 100 professeurs enseignaient leurs matières à 800 étudiants à plein temps.

En 1974, Edinburgh College of Commerce (Université de Commerce d'Edimbourg), et Napier College of Science and Technology (Université de Sciences et Technologie de Napier) ont fusionnées pour devenir Napier College of Commerce and Technology (Université de Commerce et Technologie de Napier). C'est ainsi que l'Université de Napier est devenue une institution non seulement basée sur le technique mais aussi le commerce.

En 1986, l'Université de Napier est la première institution en Ecosse à détenir une accréditation lui permettant de valider et contrôler ses propres cursus et diplômes. Et ce n'est qu'en 1992 que l'institut a pu utiliser le titre de Napier University (Université de Napier).

Enfin, en 1996, Napier agrandit encore ses capacités d'enseignement en proposant de nouveaux cursus dans les domaines de la biologie et de la santé.

C'est ainsi qu'au cours des années, l'Université a pu s'agrandir pour finalement former 4 facultés regroupant en tout 14 écoles :

- Engineering and Computing (Ingénierie et Informatique),
- Business School (Ecole de Commerce),
- Arts and Social Science (Arts et Science Sociale),
- Health and Life Sciences (Sciences de la vie et de la Santé).

Dans chaque faculté, des groupes de recherche exercent différents travaux dans des domaines variés.

1.1.2 La recherche en informatique à Napier

A chaque composante de l'Université de Napier, on trouve des groupes de recherche. Ces groupes forment une communauté de chercheurs très active. La politique est simple : à chaque département est attribué au moins un groupe de recherche. C'est ainsi que l'Université a réussi à se faire une réputation internationale dans le domaine de la recherche.

Le département Informatique possède cinq groupes de recherche :

- **Database and Object Systems** (Bases de données et Systèmes Objet)

Ce groupe est impliqué dans le développement et l'application de nouvelles bases de données et de technologies orientées objet. La plupart de ces recherches sont de type multi-disciplinaires. En base de données, les principaux domaines de travail sont la bio-informatique et l'informatique de visualisation et de transport. A titre d'exemple, des travaux ont été effectués sur la modélisation de piétons : le comportement d'individus marchant dans la rue a ainsi été recréé.

- **Evolutionary Computing** (Informatique Evolutive)

Le but de ce centre de recherche est de résoudre des problèmes de types industriels ou commerciaux : il faut dans ces domaines trouver des méthodes pour trouver des solutions acceptables dans les plus brefs délais. Il se trouve que beaucoup de systèmes organiques ou sociaux savent faire ce genre de choses. C'est donc en s'inspirant de ces systèmes que les chercheurs essaient de trouver des méthodes efficaces pour résoudre des problèmes humains.

- **Social Informatics** (Informatique Sociale)

L'informatique dite sociale crée un lien entre l'étude du design, les technologies de communication, et l'utilisation d'informations ; elle prend en compte leurs intéractions avec les contextes culturels et institutionnels.

- **Distributed Systems and Mobile Agents** (Systèmes « Distributifs » et Agents Mobiles)

Ce groupe de recherche effectue des études dans différents domaines tels que l'émulation de réseau, la sécurité, les agents mobiles, les protocoles de réseaux... etc... Il a déjà gagné plusieurs récompenses pour son travail.

- **Human Computer Interaction** (Intéraction entre Humains et Ordinateurs)

Les recherches sont basées sur de nouvelles intéractions possibles entre l'homme et la machine. Elles touchent un large panel de domaines dans lesquels l'informatique est utilisée. On trouve aussi beaucoup de recherches sur le design d'intéraction pour les handicapés : par exemple les navigateurs web pour les personnes aveugles. Les domaines du son et des intéractions tactiles font également partie de ce groupe. Il s'agit d'ailleurs des deux domaines dans lesquels s'inscrit mon projet.

1.2 Contexte du stage

Afin de comprendre le but du stage, et la motivation de mon maître de stage, Grégory Leplâtre, il est important de connaître l'évènement qui doit accueillir la suite de mon projet : NIME (New Interfaces for Musical Expression ou Nouvelles Interfaces pour l'Expression Musicale).

1.2.1 NIME : l'évènement moteur du projet

Depuis 2001, a lieu l'évènement NIME. Il s'agit en fait d'un salon où les dernières trouvailles en matière d'audiovisuel sont exposées. C'est aux différents NIME que les premières expérimentations en matière de mouvement et de musique ont été effectuées. Et c'est à ces expérimentations que mon sujet de stage est directement lié.

Le but de ces rencontres est de rassembler, puis partager les connaissances des chercheurs et des musiciens sur le travail effectué dans le domaine de la musique.

On a pu voir aux différents NIME des recherches sur l' « air guitar », ou la pratique de la guitare sans guitare, en utilisant des capteurs optiques. Evidemment, le rendu n'avait rien de très réaliste au niveau du son : il ne s'agissait là que d'expérimentations.

1.2.2 Exemples de recherches effectuées pour NIME

Mon premier objectif était de connaître le type de travail qui a été réalisé pour les précédentes éditions de NIME. Or, NIME a déjà connu de nombreux projets traitant du contrôle de sons par les gestes. Quelques travaux ont particulièrement retenu mon attention : par exemple la « GXTAR », guitare améliorée, et un système permettant d'ajouter au son d'un saxophone des effets contrôlés par les gestes du musicien.

1.2.2.1 La « GXTAR », un exemple de travail sur une guitare

Le but de la « GXTAR » est de recréer un instrument semblable à une guitare, mais ayant un fonctionnement basé sur des capteurs de position. D'une vraie guitare, les chercheurs n'ont gardé que le manche et le corps.

Ce projet m'a intéressé du point de vue utilisation de capteurs, même si ces capteurs ne sont pas du même type que ceux que j'ai utilisé pour mon projet.

Ici, deux lignes de capteurs sensibles au touché simulent deux cordes le long du manche. Ainsi, il a été possible de recréer le son d'une note, selon la position d'un doigt du musicien sur le manche. Un autre type de capteur, un joystick à 3 dimensions a permis de simuler le tirage des cordes. Ainsi, les chercheurs, en inventant un instrument, on aussi inventé une nouvelle manière d'appréhender le jeu musical. C'est sur ce point que ce projet m'a interpellé.

1.2.2.2 Des effets contrôlés par des gestes sur un saxophone

Un deuxième projet, toujours dans le même esprit, s'approche un peu plus du mien. Il s'agit d'un saxophone, non modifié cette fois ci. Par contre, une multitude de capteurs de mouvement sont placés sur le saxophoniste. A chaque capteur correspond un effet, ou un son différent. Le musicien peut alors agir comme bon lui semble sur le son produit, avec la seule aide de ses mouvements.

Evidemment, l'utilisateur peut configurer comme bon lui semble son panel d'effets selon les mouvements choisis. Ainsi, les pieds, les bras, le corps tout entier fait partie de l'instrument, participe à l'élaboration du morceau de musique.

C'est ainsi que le musicien a l'impression de jouer d'un nouvel instrument, alors qu'il ne fait qu'utiliser les mouvements de son jeu habituel. Le saxophoniste peut être assimilé à un danseur, tellement les gestes prennent de l'importance dans le morceau de musique. Grâce à ce système, il doit être beaucoup plus conscient de chacun de ses gestes qu'en temps normal.

1.3 Les conditions de travail

1.3.1 Matériel mis à disposition

Dans l'université, un laboratoire de recherche nous a été attribué, à moi ainsi qu'aux autres stagiaires qui travaillaient dans le domaine de la musique ou du son. Nous étions en tout au nombre de quatre. Dans ce laboratoire, nous disposions d'ordinateurs de marque Apple, ayant les capacités nécessaires pour remplir nos besoins. On nous a laissé le choix de travailler sur nos ordinateurs personnels ou sur les ordinateurs de l'Université. Pour ma part, j'ai choisi de travailler sur mon ordinateur personnel, pour éviter de perdre du temps dans un environnement Mac qui m'est inconnu. Nous avions également le choix d'utiliser ou non les hauts-parleurs disponibles dans le laboratoire afin de profiter d'un son de meilleur qualité.

Des capteurs de mouvement m'ont été prêtés. Il s'agit d'un matériel très coûteux, c'est la raison pour laquelle nous ne disposions que d'une seule interface Bluetooth pour les capteurs : un seul poste pouvait travailler avec les capteurs à la fois. Je ne disposais donc pas des capteurs à tout moment. Il m'a fallu m'organiser autour de cette contrainte.

1.3.2 Horaires de travail

Je travaillais du lundi au vendredi de 9h à 17h, avec une heure de pause pour le déjeuner, soit 35 heures par semaine. Il s'agit des horaires de travail appliquées à la plupart des salariés de l'Université.

1.3.3 Contacts humains

Les contacts humains ont pris une place très importante dans le projet : ils m'ont permis d'avancer avec un avis externe pour me guider. En effet, Grégory Leplâtre, mon maître de stage, me rendait visite au moins deux à trois fois par semaine. A chaque rencontre, un point d'avancement était fait. Toutes ces rencontres m'ont permis d'éviter d'éventuels écarts par rapport au projet.

J'ai également été amené à rencontrer d'autres personnes, qui, intéressées par le projet, souhaitaient en savoir un peu plus. Par exemple, Stephen Davismoon, directeur de l'école de musique de Napier, en fait partie.

Il va de soi que la langue anglaise a été utilisée tout au long du stage ; même avec Grégory Leplâtre, qui, même s'il est Français, parle anglais dans le cadre de son travail.

2 Les nouvelles notions indispensables

Ce stage, au thème bien particulier, puisqu'il s'agit de musique assistée par ordinateur, m'a obligé à me pencher sur de nouveaux concepts en matière d'informatique : la programmation dite visuelle avec la mise en oeuvre du logiciel Max/MSP, et l'utilisation de capteurs de mouvement.

2.1 Max/MSP : un logiciel pour le traitement du son

2.1.1 Présentation de Max/MSP

Max/MSP est le résultat de la fusion de deux logiciels : Max et MSP. Ces deux logiciels ayant un fonctionnement similaire et étant complémentaires, il a été décidé de les réunir. Max a été développé en 1980 par l'IRCAM[1] à Paris. En 1997, Max se voit fusionner avec MSP (Max Signal Processing). MSP n'est en fait que l'import de Pure Data, un logiciel libre de traitement du signal sonore, dans l'environnement Max. Max/MSP est un logiciel propriétaire et commercial.

Il s'agit d'un environnement graphique de programmation sonore. Il est utilisé pour le MIDI et le traitement des signaux audio. Le fonctionnement est basé sur la notion d'objets, reliés entre eux, qui se trouvent eux mêmes dans des patchers[2]. Une fois les objets disposés et reliés selon ses besoins, on obtient un algorithme graphique de traitement du son.

Il ne s'agit pas uniquement d'un logiciel de traitement du son, puisqu'il est aussi possible de réaliser des calculs, des interfaces graphiques, grâce à l'important nombre d'objets proposés. De plus, il existe un plugin nommé « Jitter » qui permet d'éditer des signaux de type vidéo.

Max est à l'origine un logiciel utilisé pour transformer des données de type MIDI[3]. Un message MIDI est composé de 4 données :

- la note : un entier correspondant à un demi-ton,
- la vélocité : un entier correspondant au volume de la note jouée,
- la durée : un entier correspondant à la longueur de la note dans le temps,
- l'instrument dans lequel est joué la note.

Toutes ces données sont des nombres entiers, ce qui explique la nécessité d'effectuer des calculs pour pouvoir traiter un message MIDI.

MSP est quant à lui un logiciel utilisé pour traiter un signal sonore. Un signal sonore n'est ni plus ni moins que le mélange d'une fréquence et d'une amplitude.

---

1 IRCAM : Institut de Recherche et Coordination Acoustique/Musique
2 Patcher Max/MSP : un patcher correspond à une fenêtre contenant des objets
3 MIDI : Musical Instrument Digital Interface

La fréquence correspond au type de son. A titre d'exemple, l'oreille humaine perçoit des fréquences situées entre 20 Hz (en dessous les sons sont qualifiés d'infrasons) et 20 kHz (au-delà les sons sont qualifiés d'ultrasons).

L'amplitude s'apparente au volume du son joué.

Ainsi :

- plus la fréquence est élevée, plus le son est aïgu,
- plus l'amplitude est élevée, plus le volume est élevé.

La courbe d'un son est donc de la forme :

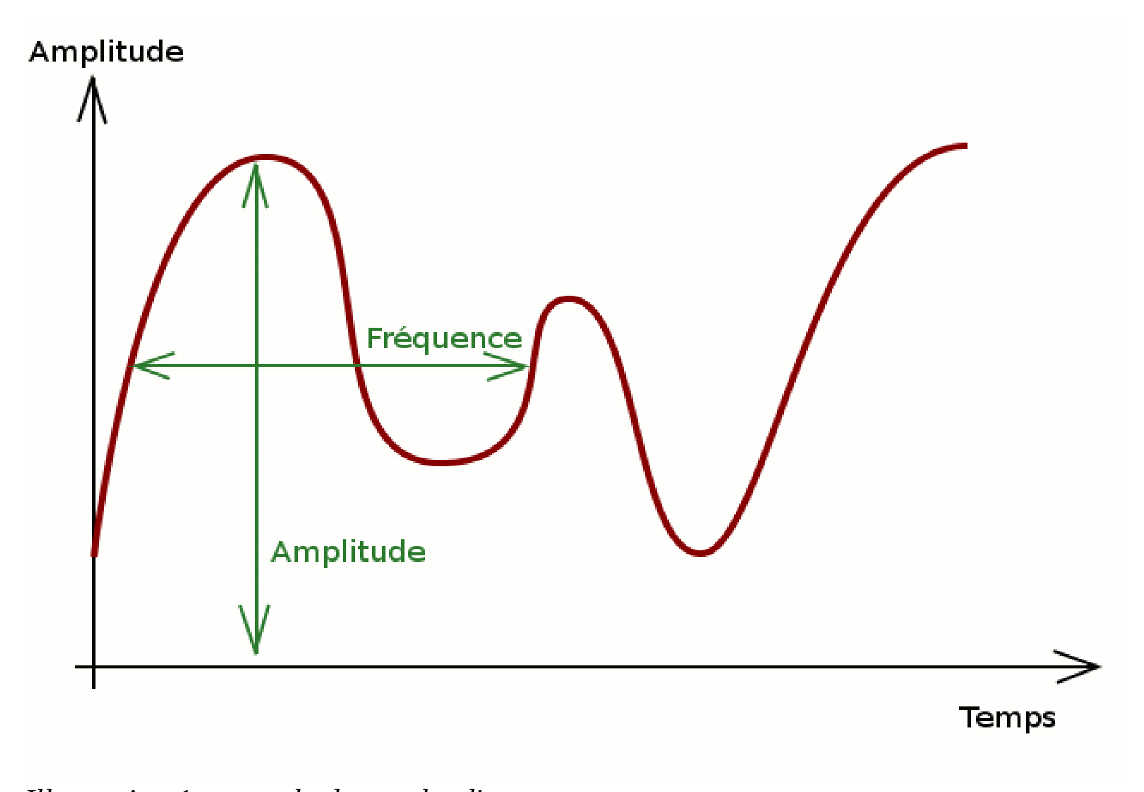


Illustration 1: exemple de courbe d'un son.

C'est pourquoi un morceau de musique peut être réduit à la variation d'une fréquence et d'une amplitude.

2.1.2 Utilisation de Max/MSP

Max/MSP fonctionne sur le principe d'objets reliés entre eux. Chaque objet possède une fonction qui lui est propre. Ces fonctions peuvent aller de la simple addition à la mise en oeuvre d'effets complexes sur un signal.

Les objets, tout comme des fonctions dans un langage de programmation plus traditionnel, possèdent des entrées et des sorties. L'objet « + » (ou addition) possède par exemple deux entrées et une sortie.

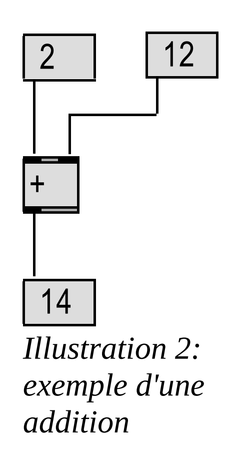


Illustration 2: exemple d'une addition

Tous ces objets sont disposés dans un patcher, qui peut être vu comme un programme à part entière. Un patcher peut contenir d'autres patchers.

Ceci étant, tout n'est pas possible avec les objets proposés. Deux solutions se présentent alors : la possibilité d'ajouter à la bibliothèque existante des objets implémentés par des programmeurs tiers, codés en langage C++. L'autre solution est d'utiliser un objet « JS » (pour JavaScript), qui permet d'appeler un fichier externe contenant un programme implémenté en JavaScript. C'est de cette manière que l'on peut facilement créer son objet Max. On peut entièrement configurer son objet : on choisit le nombre d'entrées et de sorties, les types utilisés... Ensuite en JavaScript on retrouve les notions de programmation usuelle : variables globales, boucles, fonctions... Il existe également un objet nommé « MXJ », permettant de faire appel à des classes Java. Une bibliothèque Java est aussi fournie : elle permet en premier lieu d'utiliser un objet « MXJ » et de gérer les entrées sorties.

2.1.3 Pourquoi utiliser Max/MSP ?

Max/MSP se trouve être un environnement de développement multimédia très performant, et surtout très utilisé dans le milieu de l'audiovisuel. Aux différents NIME (cf 1.2) par exemple, la majorité des produits en démonstration sont développés avec Max/MSP.

Le traitement d'un signal, qu'il soit audio, MIDI ou même vidéo, devient rapidement possible. Grâce aux objets gérant les entrées micro, ou MIDI, on peut très facilement récupérer le signal que l'on souhaite. Par exemple, un objet « notein » permet d'écouter le port MIDI de la machine : à chaque note jouée sur un clavier MIDI par exemple, on peut récupérer simultanément la note, la vélocité et le canal MIDI (ou l'instrument), soit les 3 paramètres d'un message MIDI.

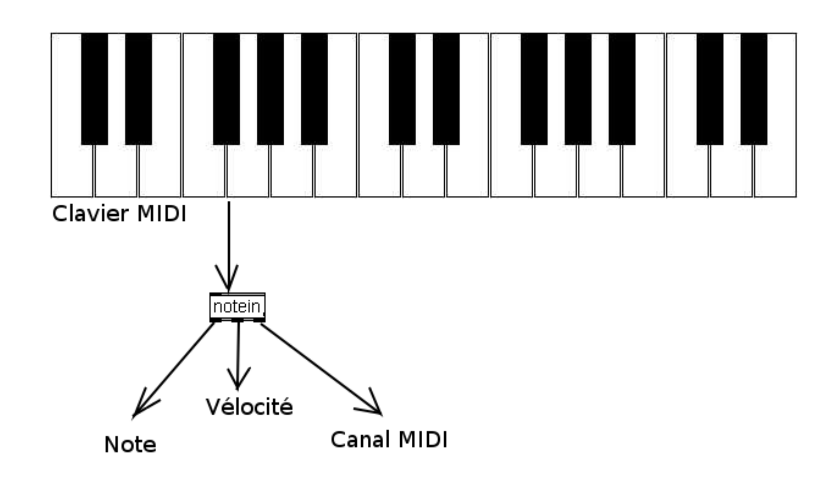


Illustration 3: exemple d'objet Max : l'objet "NOTEIN"

Il existe un objet équivalent qui, lui, va écouter, le port d'entrée audio, autrement dit le micro. Cet objet est le « ezdac ». On peut, grâce à lui, en un simple clique, décider de couper et activer le son. Cet objet a en sortie un signal audio, qui possède toutes les propriétés décrites plus haut.

Comme dit précédemment, on peut utiliser des algorithmes codés dans des langages externes à Max/MSP. Il était donc possible pour moi d'appliquer mes connaissances en programmation au traitement d'un signal audio.

Finalement, Max/MSP nous offre une interface nous permettant de jongler entre le matériel audio disponible sur une machine, et les données qu'ils renvoient. Ces données étant des chiffres, il est facile de les récupérer pour effectuer d'éventuels traitements dessus, puis de les renvoyer à nouveau vers le matériel audio.

Typiquement, un patcher Max/MSP, qui nous permettrait d'effectuer des traitements sur un signal sonore reçu, et de le renvoyer par la suite est de cette forme :



Traitement à effectuer sur le signal



Illustration 4: récupérer et jouer un signal sonore avec Max/MSP

2.2 Utilisation de capteurs de mouvements

2.2.1 Présentation

Afin de mener mon stage à bien , ainsi que celui de Justin Huss, l'Université de Napier a dû se munir de capteurs de mouvements. Différents types de capteurs nous ont alors été proposés : des capteurs d'accélération, de rotation, et de torsion.

Après avoir analysé chacun d'eux, j'ai décidé d'utiliser les capteurs d'accélération et d'inclinaison. En fait, ces deux fonctionnalités sont réunies dans un même objet. Nous disposions de deux capteurs d'accélération/rotation.

Les capteurs bénéficient d'une interface Bluetooth, permettant de supprimer tout câble entre la machine qui effectue les traitements et le musicien. Ce détail s'avèrera très pratique une fois l'application mise en oeuvre. Un logiciel fourni permet à l'ordinateur de les assimiler à un contrôleur MIDI. Ainsi j'ai pu facilement récupérer dans Max les valeurs renvoyées par les capteurs, à savoir des entiers.

La première étape pour utiliser ces capteurs était d'analyser leurs comportements.

2.2.2 Comportement d'un capteur d'accélération



Illustration 5: un capteur GForce3D de marque I-CubeX – source : infusionsystems.com

Chaque capteur d'accélération renvoie 3 valeurs, correspondant aux différents axes. Ils peuvent capter une accélération allant jusqu'à 3 G[4].

C'est la raison pour laquelle on peut dire que je disposais de capteurs à 3 dimensions. En théorie, et d'après le constructeur, les capteurs doivent renvoyer des données comprises entre 17 et 68, 42 étant l'équivalent d'une non accélération. En fait, la valeur 68 correspond à une accélération de 3G et 17 à une décélération de 3G.

Premier constat : les valeurs ne sont pas les mêmes selon les axes. Par exemple, lorsque le capteur est posé, qu'il ne bouge pas, les 3 valeurs sont différentes, on obtient des valeurs aléatoires du type :

- **axe x** : 42,
- **axe y** : 44,
- **axe z** : 39.

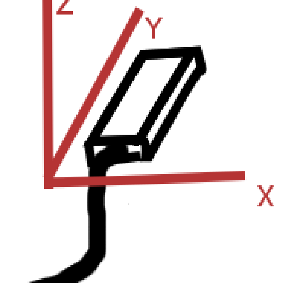


Illustration 6: les différents axes considérés par un capteur GForce 3D

4 Un G correspond à la gravité de la Terre. A Paris la valeur d'un G est de 9, 81 m/s².

Ceci est dû au fait que la rotation est gérée par ces mêmes capteurs : la rotation agit sur les valeurs renvoyées en position arrêtée, nous en parlerons plus loin.

Il est important de comprendre comment réagissent les capteurs sur un mouvement basique. Prenons par exemple un mobile qui tombe sur le sol. Typiquement, si un capteur était placé sur ce mobile, les valeurs renvoyées ressembleraient à ceci :

Instant t	Valeur renvoyée	Phase du mouvement
1	42	arrêt
2	41	accélération
3	40	accélération
4	39	accélération
5	38	accélération
6	39	accélération
7	40	accélération
8	41	accélération
9	42	mouvement rectiligne uniforme : pas d'accélération
10	42	
11	42	
12	55	forte décélération
13	42	arrêt

Phase d'accélération

Accélération nulle

Décélération

On note que le capteur réagit différemment selon le sens du mouvement. Effectivement, si on le bouge vers le haut (en suivant un axe particulier), alors les valeurs vont augmenter dans un premier temps. Si on effectue un mouvement similaire mais vers le bas, alors les valeurs vont d'abord diminuer. On peut grâce à ce mécanisme identifier le sens des mouvements.

On obtient deux évolutions des valeurs différentes selon le sens du mouvement :

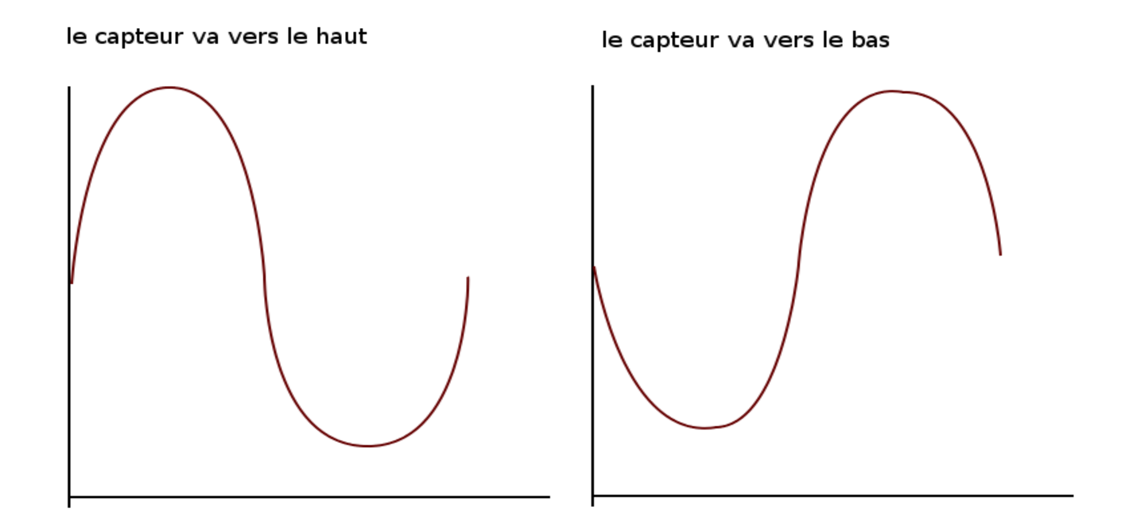


Illustration 7: évolution des valeurs selon le sens du mouvement

On peut diviser le mouvement en 3 parties :

- d'abord **l'accélération** est **positive,**
  - si le capteur va vers le **bas,** les valeurs **diminuent,**
  - si le capteur va vers le **haut,** les valeurs **augmentent ;**

- ensuite **l'accélération** est **négative,**
  - si le capteur va vers le **bas,** les valeurs **augmentent,**
  - si le capteur va vers le **haut,** les valeurs **diminuent ;**

- enfin l'accélération est **nulle,**
  - les valeurs se **stabilisent** à la valeur initiale (normalement 42).

2.2.3 Capteurs de rotation

Les capteurs de rotation utilisés sont donc les mêmes que les capteurs de translation. On ne peut se servir de leurs propriétés sur la rotation qu'en position arrêtée. Les valeurs sont renvoyées sur les mêmes sorties que lorsqu'on les utilise en « mode » translation, ce qui rend la rotation invisible si une translation est effectuée au même instant.

Comme pour la translation, trois valeurs sont renvoyées en même temps. On peut ainsi savoir autour de quel axe tourne le capteur.

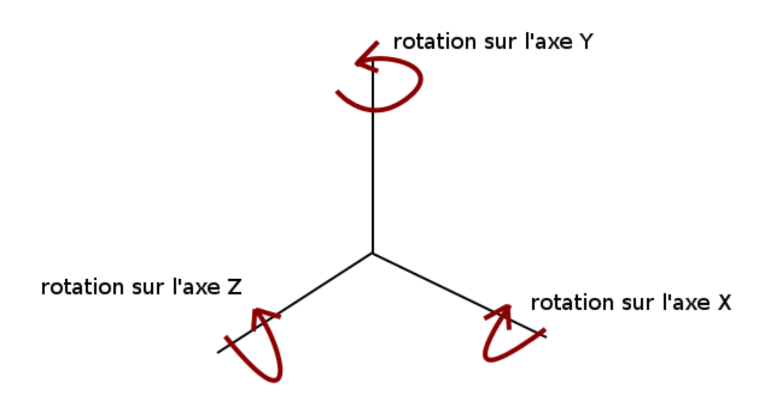


Illustration 8: Les rotations exploitables

Pour une rotation de 180°, on obtient des valeurs allant de 35 à 49, soit 42 ± 7. Le problème rencontré ici, pour l'exploitation de cette propriété des capteurs est le suivant : il est impossible d'exploiter ces valeurs autrement qu'en position arrêtée.

2.3 Les mouvements des musiciens

Tout les musiciens, quel que soit l'instrument joué, bougent selon la musique. Ces mouvements peuvent être de différents types : utiles, accompagnants, ou figuratifs.

2.3.1 Les mouvements utiles

Les mouvements dits utiles ou efficaces correspondent aux mouvements obligatoires pour produire des sons. Ce sont des gestes propres à l'instrument joué. Par exemple, un pianiste doit appuyer sur les touches de son clavier pour que son piano produise un son.

En guitare, les mouvements sont propres aux techniques utilisées. Effectivement, il existe en guitare une multitude de techniques allant de la buté (un pincement des cordes avec le pouce, de la note la plus grave à la plus aïgue), au picking (tous les doigts sont indépendants les uns des autres au niveau rythmique et mélodique), en passant par l'arpège (pincement régulier des cordes les unes après les autres). On peut aussi jouer à l'aide d'un médiator, qui autorise une plus grande amplification du son et permet un jeu rapide et brillant.

Il existe trois types de mouvements efficaces :

- les gestes **d'excitation :** ils transmettent l'énergie émise par le musicien. Par exemple, la force avec la quelle le guitariste va pincer ses cordes ;
- les gestes de **modification :** leur but est de modifier le son initialement émis par l'instrument. Par exemple, l'utilisation d'un vibrato en guitare ;
- les gestes de **sélection :** ce sont les différents mouvements des doigts correspondants aux accords, aux notes jouées.

2.3.2 Les mouvements accompagnants

Les mouvements accompagnants sont quant à eux des gestes ajoutés au jeu usuel. Ils se présentent généralement sous la forme de balancements du corps, ou de battements du pied, suivant le rythme de la musique. On voit aussi des musiciens incliner leurs instruments selon le son émis.

Les mouvements dits accompagnants sont en fait associés aux mouvements efficaces. Les deux sont liés par le son et le rythme de la musique.

2.3.3 Les mouvements figuratifs

Les mouvements figuratifs ou soniques sont des mouvements perçus par l'audience, mais qui n'ont pas de correspondance évidente avec des mouvements en particulier : ils peuvent être des gestes involontaires du musicien. Ils modifient le son produit, mais restent non maîtrisés et sont inexploitables.

3 Elaboration d'une application musicale

Dans cette partie, je vais vous présenter le travail qui a été réalisé afin de rendre l'application utlisable. Je vais donc expliquer les effets utilisés et leurs mises en oeuvre ; je vais commenter un algorithme utilisé pour la récupération de mouvements du type allés-retours ; et enfin je vais détailler l'utilité de l'interface graphique implémentée en Java.

3.1 Implémentation d'effets

Le premier travail à effectuer, une fois le logiciel Max/MSP maîtrisé, était de trouver des effets paramétrables à appliquer sur un jeu de guitare. Il est important que ces effets soient paramétrables pour que plus tard, ils soient tous gérés par les mouvements du musicien.

3.1.1 Wah-Wah

3.1.1.1 Principe du wah-wah



Illustration 9: exemple d'une pédale Wah-Wah – source : Wikipedia.org

D'habitude l'effet wah-wah est utilisé à l'aide d'une pédale dont la position est réglable. Le guitariste peut agir dessus à tout moment. On appelle cet effet le « wah-wah » pour la simple et bonne raison que lorsque le guitariste agit sur la pédale, la guitare semble prononcer des « wah ». Généralement, cet effet est utilisé dans des styles de musique tels que le funk, le jazz ou encore le reggae.

Afin de comprendre comment fonctionne un tel effet, il faut avant tout savoir sur quoi il agit. Tout d'abord, la fréquence de coupure est la fréquence à laquelle le volume de sortie est réduit à 71% du volume d'entrée.

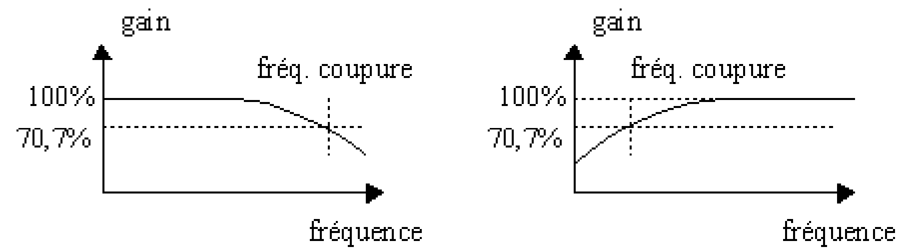


Illustration 10: Fréquence de coupure – source : Wikipedia.org

Un filtre passe-bas est un filtre qui laisse passer les basses fréquences et qui atténue les hautes fréquences, c'est-à-dire les fréquences supérieures à la fréquence de coupure. Il accentue légèrement les fréquences situées juste avant la fréquence de coupure, dans le but de faire ressortir l'effet. Ce pic d'amplification est appelé la résonance.

Une pédale wah-wah agit sur le son émis de la manière suivante : le signal sonore passe par un filtre passe-bas et la pédale agit sur la fréquence de coupure.

3.1.1.2 Mise en place avec Max/MSP

C'est donc un fonctionnement similaire à celui décrit ci-dessus que j'ai dû recréer pour mener à bien l'utilisation de l'effet wah-wah. Il existe dans Max/MSP un objet permettant de recréer un filtre passe-bas : le « filtergraph ». Il est même possible de modifier la fréquence de coupure à l'aide d'un paramètre d'entrée : c'est en modifiant sa valeur que l'on entend le « wah » créé. Ici, ce paramètre est modifié à l'aide du « slider » horizontal. Voici par exemple deux états différents de la fréquence de coupure : on note l'état du slider et celui du graphique qui changent.

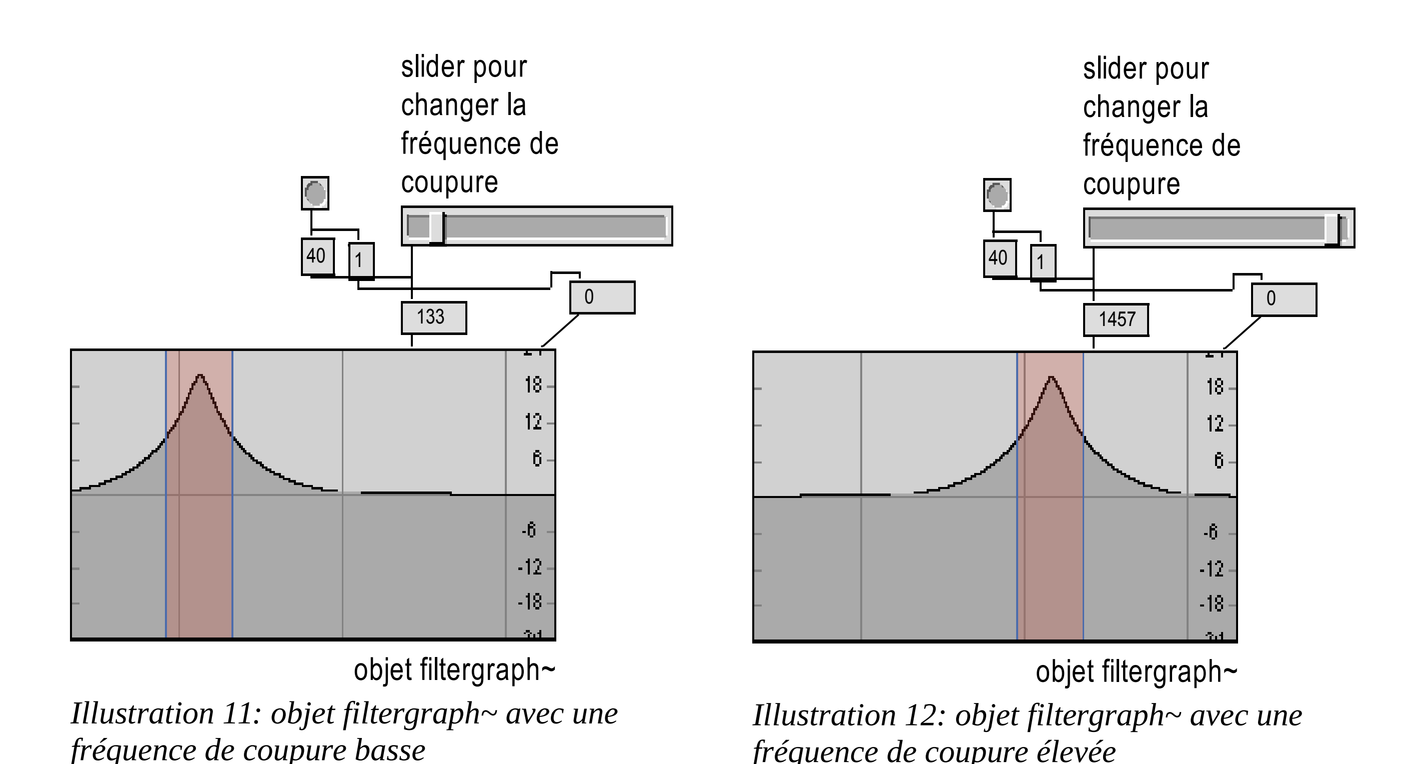


Illustration 11: objet filtergraph~ avec une fréquence de coupure basse

Illustration 12: objet filtergraph~ avec une fréquence de coupure élevée

L'algorithme qui liera le capteur de mouvement à ce patcher devra modifier la valeur d'entrée du slider en fonction des valeurs renvoyées par le capteur.

3.1.2 Délai

3.1.2.1 Principe du délai

Le délai est un effet très utilisé en musique. Son principe est simple : il doit recréer à intervalle de temps régulier un son qui a été joué précédemment. Ainsi, un « bip » qui arriverait en entrée, doit être rejoué plusieurs fois, avec une amplitude diminuant au fur et à mesure. Graphiquement, le bip joué ressemblerait à ceci :

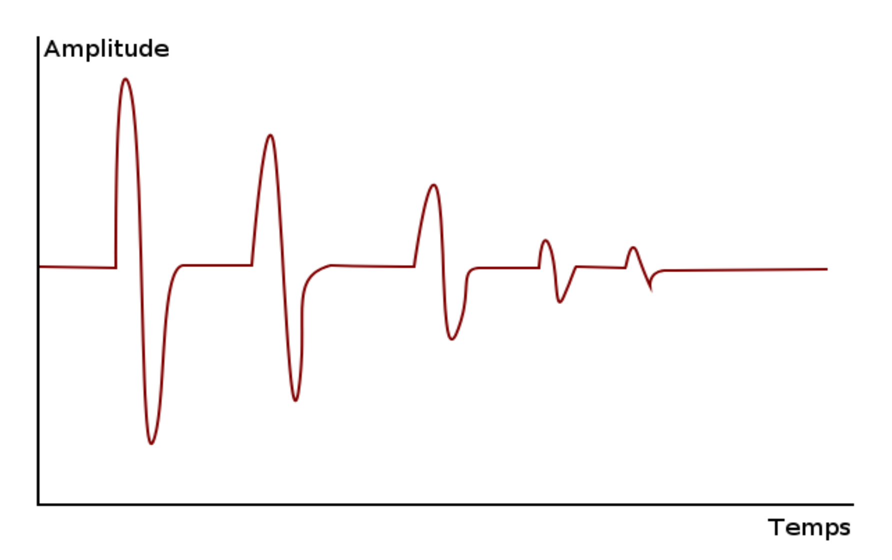


Illustration 13: schématisation d'un délai appliqué à un signal

On retrouve généralement un effet proche de l'écho.

3.1.2.2 Mise en place du délai avec Max/MSP

Pour mettre en place un délai, il faut pouvoir enregistrer le son qui arrive en entrée pendant un certain lapse de temps. Cette période d'enregistrement est l'un des paramètres de l'effet. Il faut également pouvoir rejouer ce signal après une certaine période.

Or, les objets « tapin~ » et « tapout~ » remplissent ces fonctions : l'objet « tapin~ » enregistre le son en entrée pendant le temps qu'on lui place en paramètre, et l'objet « tapout~ » ressort ce même son après le délai qu'on lui place également en paramètre.

Il s'agit par la suite de boucler ce signal sur lui même, afin qu'il repasse par ces deux objets indéfiniement. Mais si on ne réduit pas l'amplitude du signal à chaque itération, l'écho ne s'arrêtera jamais, et on entendra le même son en boucle. C'est pourquoi, à la sortie de la boucle, on décide de multiplier l'amplitude du signal par une valeur comprise entre 0 et 1, par exemple 0,5 : à chaque itération, le volume du signal va être divisé par deux. Ce multiplicateur est un paramètre de plus.

Voici un extrait du patcher permettant de reproduire un effet de délai : j'ai choisi d'attribuer aux objets « tapin~ » et « tapout~ » la même valeur, ce système fonctionnant bien.

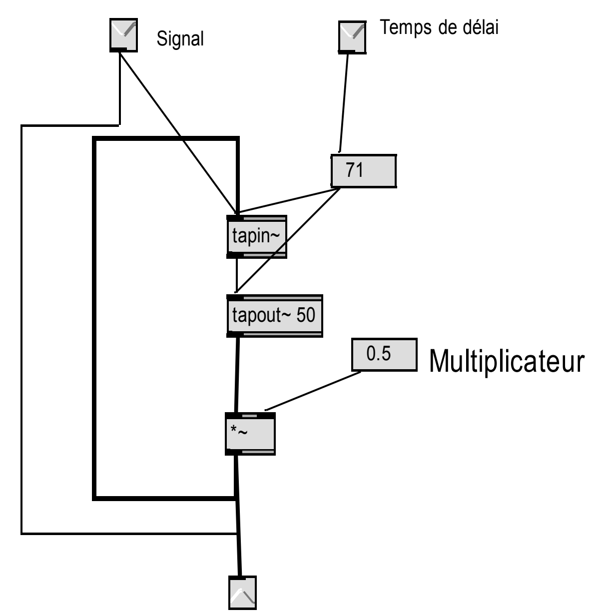


Illustration 14: patcher Max/MSP créant un effet de type "délai"

3.1.3 Larsen

3.1.3.1 Présentation du larsen

Le larsen, aussi appelé « feedback », repose sur le même principe que le délai. Un signal effectue une boucle sans fin. L'unique différence avec le délai est le fait que ce signal ne soit pas atténué à chaque itération. Les guitaristes utilisent souvent cet effet en plaçant leur guitare face à leur amplificateur : le signal effectue alors une boucle sans fin de la guitare à l'amplificateur puis de l'amplificateur à la guitare. Souvent, les larsens sont ressentis comme des sifflements ou des bourdonnements dont le volume augmente sans cesse. Ceci est dû au fait que l'amplitude augmente à chaque itération.

3.1.3.2 Mise en place du larsen avec Max/MSP

Pour obtenir un larsen, il faut que le temps de délai soit très court, afin que l'on n'entende pas un effet similaire au délai. La deuxième condition pour obtenir un larsen, la plus importante, est la valeur du multiplicateur du délai : elle doit être à 1. Dans la pratique, elle ne peut pas être à 1 : l'amplitude augmentant sans cesse, le son est coupé dans Max/MSP afin de sauvegarder le matériel. J'ai donc choisi de placer cette valeur à 0.98. Ainsi, l'effet est audible, et l'amplitude n'augmente que de manière raisonnable.

3.1.4 Vibrato

3.1.4.1 Présentation du vibrato

Le vibrato est un effet très utilisé en guitare, on le trouve sous deux formes : il peut être créé soit en tirant la corde avec le doigt sur le manche, soit en utilisant un accessoire installé sur la guitare. Le principe est de provoquer une variation rapide de la hauteur du son autour de sa tonalité. Bref, la fréquence de vibration est modifiée très rapidement autour du son d'origine.

3.1.4.2 Mise en place du vibrato dans Max/MSP

Pour reproduire un vibrato dans Max/MSP, on utilise un filtre de peigne (« comb filter » en anglais). Le filtre de peigne porte ce nom car il transforme le signal en lui modifiant sa fréquence, ce qui donne une représentation graphique du signal ressemblant à un peigne. Pour un signal lambda, on obtient un graphique de ce style :

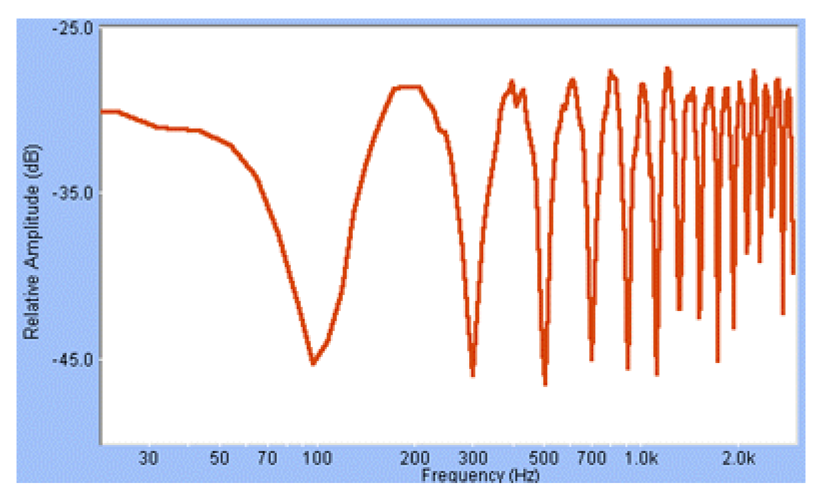


Illustration 15: résultat du filtre de peigne - source : espace-cubase.org

Dans Max/MSP, il s'agit de modifier le paramètre délai d'un objet « comb~ » pour obtenir cette variation de fréquence, et donc l'effet vibrato. Notons qu'un gain doit être initialisé (à 0.88 dans l'exemple ci-contre) afin que l'objet soit prédisposé à générer un signal audible.

Nous retrouvons, comme pour l'effet wah-wah, un slider sur lequel il faut agir pour que l'effet se fasse entendre. C'est pourquoi, une fois encore, l'algorithme qui liera le capteur de mouvement à ce patcher devra modifier la valeur d'entrée du slider en fonction des valeurs renvoyées par le capteur.

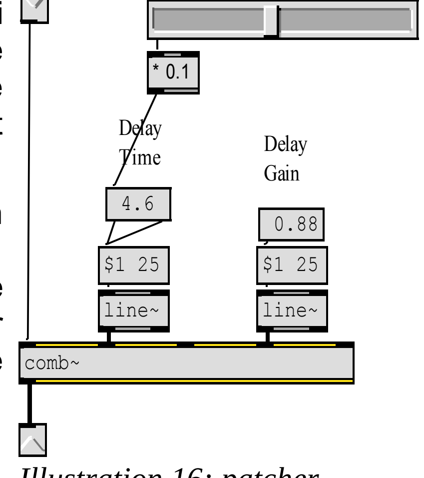


Illustration 16: patcher Max/MSP recréant l'effet d'un vibrato

3.2 Un exemple d'algorithme exploitant des mouvements

Pour chaque effet, différents algorithmes ont été mis en oeuvre. Certains contrôlent les angles, d'autres les mouvements.

J'ai décidé de commenter un algorithme permettant d'exploiter les mouvements de type allés-retours. Il permet différencier le sens du mouvement (vers le haut ou vers le bas) selon un axe. Cet algorithme est écrit en utilisant le langage JavaScript, et est adapté au compilateur JavaScript intégré dans Max/MSP. C'est la raison pour laquelle certaines fonctions sont propres à l'utilisation de JavaScript dans le logiciel.

Rappel : les algorithmes sont intégrés aux patchers Max/MSP grâce à l'objet « js », prenant en paramètre le nom du fichier JavaScript.

L'algorithme est appelé à chaque événement se produisant en entrée de l'objet « js ». En l'occurence, ici, il sera appelé à chaque changement de valeur venant de « ctlin 6 ».

ctlin 6
0
js sensorVibrato.js
0

```
inlets=2;
outlets=1;
```

Ces deux variables globales définissent le nombre d'entrées et de sorties requises par l'objet « js » résultant de cet algorithme dans Max/MSP.

```
var initialAcc=50;        // acceleration initiale
var curPos=50;            // position courante initialisée à 50
var curAcc;               // acceleration courante
var value;
var testWay=0;            // variable permettant de connaître
                          // l'étape du mouvement
var sign;
var sensitivity=1   ;     // sensibilité du capteur
var limitMin=0;           // valeur minimum de sortie
var limitMax=100;         // valeur maximum de sortie
var multiplier=5;         // pas
```

Comme dans tout algorithme, suit la déclaration des variables. Il s'agit ici de variables globales : leurs valeurs sont sauvegardées en mémoire entre deux appels.

La variable testWay permet de savoir à quel étape du mouvement on se trouve à un moment donné : elle prend :

- la valeur 0 si le mouvement est arrêté,
- la valeur 1 si le mouvement est dans la première phase (l'accélération),
- la valeur 2 si le mouvement est dans la seconde phase (la décélération).

```
function msg_int(v)
{
	var acc;
	curAcc=v-initialAcc;
	acc=curAcc;
```

La fonction msg_int est la fonction qui est appelée lorsque de nouvelles valeurs sont entrées. Ici, c'est la fonction sur laquelle repose tout le bon fonctionnement de l'algorithme : elle est la seule fonction qui sera appelée. La variable v, passée en paramètre de la fonction, correspond à la valeur envoyée par le capteur.

```
	if(acc < 0) {
		acc=acc + (2*acc);
	}
```

On attribue à la variable acc sa valeur absolue : il n'existe pas en JavaScript de fonction permettant d'avoir directement la valeur absolue...

```
	if (testWay==0)     // c'est le début du mouvement
	{
		if (v<initialAcc-sensitivity ||
v>initialAcc+sensitivity)
		{
			if (curAcc>0)  // le capteur va vers le haut
			{
				setSlider(+1); // appel à la fonction
				               // setSlider définie plus bas
				testWay=1;
				sign="pos";
			}
```

On entre dans le premier if si le mouvement était arrêté à l'étape précédente. Puis on entre dans le second if si le mouvement détecté est assez important (d'où la variable gérant la sensibilité) : le suivi du mouvement se déclenche alors.

Le but de l'algorithme est de savoir à tout moment où en est le mouvement. C'est pourquoi on met la variable testWay à l'état 1 (équivalent à la première phase).

On indique par la variable pos que curAcc est positive, donc que le mouvement est vers le haut. On se servira de cette information plus tard.

```
        if (curAcc<0)   // le capteur va vers le bas
                {
                        setSlider(-1);
                        testWay=1;
                        sign="neg";
                }
        }
}
```

On applique le même principe si curAcc est négative, ou que le capteur va vers le bas.

```
if (testWay==1)//on est dans la phase d'accélération
{
        if (curAcc>0)    // le capteur va vers le haut... ou pas !
        {
                if (sign=="neg") // le signe a changé, la phase
                             // a changé, le capteur va vers
                        // le bas
                {
                        setSlider(-1);
                        sign="pos";
                        testWay=2;
                }
                else  // le capteur va toujours vers le haut
                {
                        setSlider(+1);
                }
        }
```

On est dans la phase d'accélération. Si curAcc est positive, alors c'est que le capteur va vers le haut, sauf dans le cas où le signe (variable sign) a changé. La variable sign est en fait un mouchard, permettant de connaître le signe de l'accélération à l'étape précédente. Si le signe change, alors le mouvement entre dans la phase 2, la phase de décération.

```
        if (curAcc<0) // le capteur va vers le bas... ou pas !
        {
             if (sign=="pos")    // le signe a changé, la phase
                                 // a changé, le capteur va vers
                                 // le haut
             {
                  setSlider(+1);
                  sign="neg";
                  testWay=2;
             }
             else // le capteur va toujours vers le bas
             {
                  post("testWay = 1 down \n");
                  setSlider(-1);
             }
        }
   }
```

On retrouve le principe analogue à celui décrit précédemment, mais dans le cas où le capteur va vers le bas.

```
	if (testWay==2)		// on entre dans la phase de décélération
	{
		if (curAcc>0)  // le capteur va vers le bas
		{
			setSlider(-1);
		}

		if (curAcc<0)  // le capteur va vers le haut
		{
			setSlider(+1);
		}

		if(curAcc==0)  // le mouvement est
		               // terminé
		{
			testWay=0;
			curPos=50;
		}
	}
```

Il s'agit toujours du même principe dans la deuxième phase, mais les valeurs renvoyées sont inversées par rapport à la phase 1 : si curAcc est positive, on renvoie une valeur diminuée et inversement.

Si la valeur de curAcc est à zéro, alors on considère le mouvement comme terminé. On remet testWay à 0 et curPos à 50.

```
	value=v;
	outlet(0,curPos);
}
```

Cette dernière partie est appelée à chaque fois que la fonction msg_int est appelée. La variable value sert à l'initialisation (fonction initialize, voir plus bas). On renvoie la valeur de curPos sur la sortie de l'objet « js » à l'aide de la fonction « outlet ».

```
function setSlider(v)
{
	if (v==+1)
	{
		if (curPos < limitMax)
		{
			curPos=curPos+multiplier;
		}
	}
	if (v==-1)
	{
		if (curPos > limitMin)
		{
			curPos=curPos-multiplier;
		}
	}
}
```

La fonction setSlider est la fonction appelée pour augmenter ou diminuer la valeur renvoyée dans la fonction msg_int.

```
function initialize()
{
	initialAcc=value;
	curPos=50;
	testWay=0;
}
```

On utilise la fonction initialize pour définir la position initiale du capteur. Je rappelle que les capteurs prenant en compte la rotation, la valeur renvoyée en position arrêtée n'est que rarement la même selon la position du capteur. Cette fonction permet de régler un soucis de calibration du capteur.

3.3 Création d'une interface graphique

3.3.1 Une interface homme-machine implémentée en Java

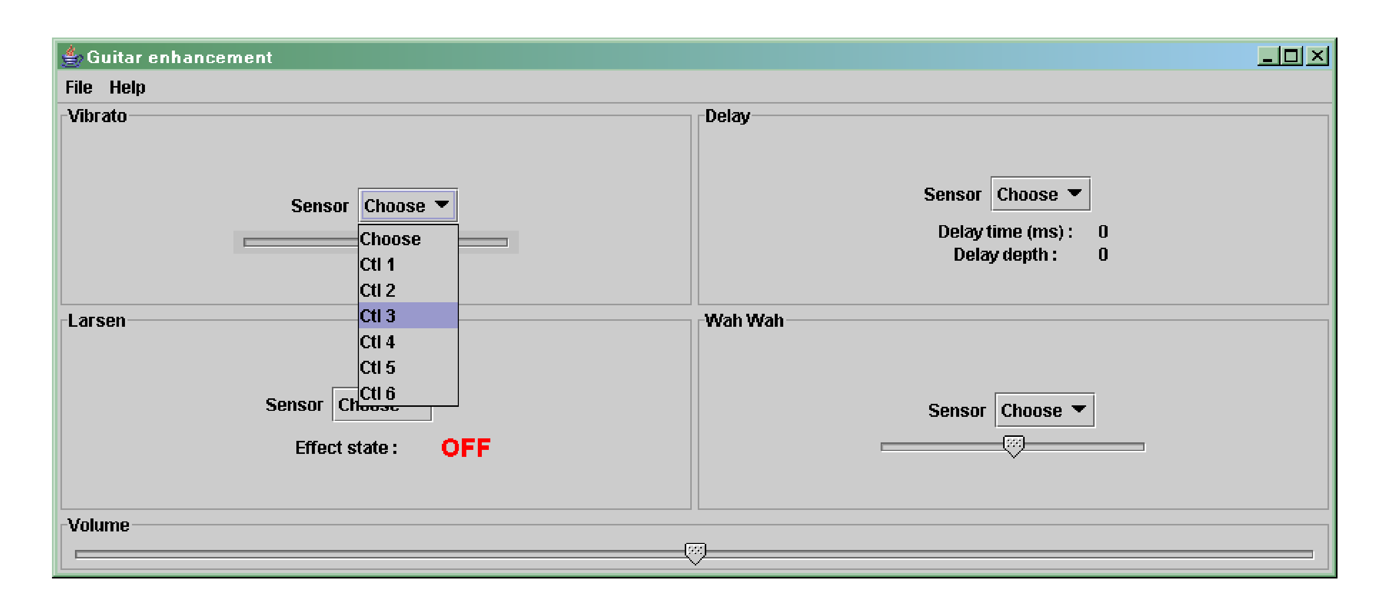
Il est possible d'intégrer des classes Java à Max/MSP en suivant le même principe que l'intégration de JavaScript. Un objet « mxj » analogue à « js » permet de faire cela. Il s'est avéré impossible de tout implémenter en Java, les algorithmes devant répondre en temps réel pour obtenir un rendu sonore acceptable ; en effet, Java est trop lent pour ce faire.

C'est donc uniquement l'interface graphique que j'ai implémenté en Java. Ce pour deux raisons : la première, il est facile d'intégrer une interface graphique Java à un patcher Max/MSP ; la seconde, Java est le seul langage dans lequel je suis capable d'éditer une interface graphique.

Cette interface graphique a pour but de rendre l'outil développé plus attrayant pour un utilisateur lambda. L'application étant avant tout destinée à des musiciens, il est important qu'ils puissent facilement la configurer. Ainsi, à chaque capteur, ou plus exactement à chaque axe de capteur, le musicien peut assigner un effet en particulier. C'est grâce à cette assignation qu'on peut dire que l'application développée ici est configurable.

Il existe une bibliothèque nommée MaxObject, qu'il suffit de placer en extension de la classe appelée par Max/MSP. Grâce à elle, on peut implémenter la méthode « loadbang() », qui n'est autre que la méthode appelée lors du démarrage du patcher, dans laquelle on instancie une interface graphique. Ensuite, les relations entre interface graphique et patcher Max/MSP se font par utilisation d'inlets et outlets : les entrées et sorties de l'objet « mxj ».

3.3.2 Un aperçu de l'IHM



Ici, l'utilisateur peut choisir d'assigner à chaque effet un capteur. On rappelle que chaque capteur renvoie trois valeurs. On reprend ici la terminologie du logiciel qui gère les capteurs en amont. Ainsi, l'utilisateur retrouve les expressions « ctl » avec lesquels il est déjà familier.

On remarque que l'application est créée pour l'utilisation simultanée de deux capteurs maximum (on n'a que 6 contrôleurs disponibles, 3 par capteur).

En plus de l'assignation des capteurs à des effets, l'interface permet de visualiser l'état de chaque effet : il est toujours pratique, lorsque l'on joue de la guitare, de savoir où en sont les effets à un instant donné. Les sliders utilisés ici ne peuvent pas être modifiés par l'utilisateur : seuls les mouvements des capteurs agissent sur eux.

Ensuite, le musicien a la possiblité de contrôler le volume de sortie du son, à l'aide du slider au bas de la fenêtre.

Conclusion

Au cours de ce stage, il m'a été indispensable d'apprendre à me servir d'un nouveau logiciel : Max/MSP. Cette tâche m'a été facile puisque les notions pour l'utiliser ne m'étaient pas inconnues étant donné qu'il s'agit de programmation, bien que visuelle. J'ai tout de même dû apprendre quelques fondamentaux en matière de traitement d'un signal sonore. Il m'a quand même été indispensable d'utiliser des langages de programmation plus habituels tels que JavaScript et Java, dans lesquels tous les algorithmes et l'interface graphique sont écrits. Un travail de recherche a également été réalisé pour comprendre et mettre en place des effets sonores maîtrisables.

C'est à la suite de l'étude effectuée autour du contexte, du comportement des capteurs, mais aussi des mouvements des guitaristes, que j'ai pu réaliser mon application. Il se trouve qu'elle n'est que l'origine d'un projet qui sera continué par la suite. En effet, le groupe de recherche va poursuivre l'étude pour obtenir une application plus évoluée. Cette-dernière, si tout se passe comme convenu, sera présentée au NIME 2008.

J'ai, grâce à ce stage, pu découvrir l'environnement de travail dans une Université anglo-saxone. J'ai travaillé de manière autonome, bien que guidé par Grégory Leplâtre, mon maître de stage, lorsque j'en avais besoin. Ce stage était avant tout pour moi l'occasion de travailler dans un domaine qui m'intéresse : j'ai pu appliquer mes connaissances en programmation à un domaine bien particulier : la musique. Même si je ne pense pas continuer plus tard dans le milieu de la recherche et de la musique, cette expérience m'aura appris à être autonome et à travailler avec méthode, en suivant une démarche à laquelle je n'étais jusqu'alors pas habitué.

Index des illustrations

Sources documentaires

Tutoriaux de Max/MSP (auteur : Cycling' 74)

Site Internet de Infusion Systems (constructeur des capteurs de mouvement)
infusionsystems.org

Encyclopédie en ligne Wikipédia
fr.wikipedia.org
en.wikipedia.org

Site Internet de l'Université de Napier
napier.ac.uk